METHODS AND APPARATUS FOR DATABASE TRANSACTION QUEUING

FIELD OF THE INVENTION

[0001] The present invention relates to methods and apparatus for orderly access to a database or database server, and more particularly to methods and apparatus for queuing of database transactions for a transaction system having multiple transaction service threads.

BACKGROUND OF THE INVENTION

[0002] In some database systems, users enter transactions to manipulate the contents of the database. Each transaction is prepared by a user, which requires that users handle the overhead of constructing and submitting the transactions. This overhead varies depending upon which database interface is being used, and can range from non-existent or minimal to significant. At least one known database system includes a database abstraction layer that has a moderate level of overhead associated with it. This database system is built upon a Java® JDBC interface and requires that each separate transaction on the database take place in a thread within a separate Java ThreadGroup. Setting up a transaction involves designating an appropriate Thread within a ThreadGroup or starting Thread within an appropriate Threadgroup, signaling the beginning of a transaction and handling any error conditions from possible failures to get a correct context and/or from network failures, and clearing a cache of a connection if it has been used previously to prevent the possibility of reading out-of-date values from the cache. In addition to the set up overhead, there is also overhead

associated with handling of failed transactions. This overhead includes committing the transactions after all database changes are completed, dealing with possible errors by writing to a log file and notifying interested listeners, and rolling back any changes if necessary. In addition, transactions are processed as the users submit them. Thus, it is possible to overwhelm the database with a large quantity of nearly simultaneous transactions.

SUMMARY OF THE INVENTION

[0003] To better manage the resources of databases and to avoid overwhelming databases with large numbers of nearly simultaneous transactions, one configuration of the present invention provides a method for optimizing database transaction performance in a database transaction processor having transaction services threads capable of being in active, non-active, and waiting states. The method includes: (a) adding a database change to a top of a queue; and (b) starting a non-active transaction service thread conditioned upon less than a predetermined maximum number of transaction service threads being present.

[0004] Another configuration of the present invention provides a computing apparatus having a central processing unit operatively coupled to a memory including a database change queue, the apparatus configured to process a plurality of treads capable of being in active, non-active, and waiting states, the apparatus further configured to: (a) add a database change to a top of the database change queue; and (b) add a non-active transaction service thread, or change a waiting transaction service thread to a non-active state, conditioned upon whether there are less than, or not less than a

predetermined maximum number of transaction service threads present, respectively.

[0005] In yet another configuration of the present invention, there is provided A machine-readable medium or media having recorded thereon instructions configured to instruct a computing apparatus having a central processing unit operatively coupled to a memory to: (a) add a database change to a top of the database change queue in the memory; and (b) start a transaction service thread in a non-active state, or change an existing transaction service thread in a waiting state to a non-active state, conditioned upon whether there are less than, or not less than a predetermined maximum number of transaction service threads present, respectively.

[0006] Further areas of applicability of the present invention will become apparent from the detailed description provided hereinafter. It should be understood that the detailed description and specific examples, while indicating the preferred embodiment of the invention, are intended for purposes of illustration only and are not intended to limit the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The present invention will become more fully understood from the detailed description and the accompanying drawings, wherein:

[0008] Figure 1 is a flow chart representing the queuing of database changes and control of transaction service threads in one configuration of the present invention.

[0009] Figure 2 is a flow chart representing a transaction service thread that processes database changes in one configuration of the present invention.

[0010] Figure 3 is a representation of a computing apparatus and a medium or media of one configuration of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0011] The following description of the preferred embodiment(s) is merely exemplary in nature and is in no way intended to limit the invention, its application, or uses.

[0012] In one embodiment and referring to Figure 1, database changes are added to a queue and handled by a thread running in a computing apparatus. More particularly, in configuration 10, when a database change is requested, the database change is added 12 to the top of a queue in a memory of the computing apparatus. Depending upon the database change, there may be one or more listeners associated with the change. "Listeners" are programming objects that are notified when the associated change is begun. Listeners are also notified of the completion status of the associated change (e.g., "successful completion," or "failure") upon completion of the transaction.

[0013] Transactions to the database are performed utilizing "transaction service threads." A "transaction service thread" is a programming thread executed by a central processing unit of the computing apparatus. Transaction service threads can be in an active state, a non-active state, or a waiting state. An active transaction service thread is a thread that is in the

process of handling a set of database changes specified by a queued database change. A non-active thread is a thread that is in the process of attempting to service the queue of database changes. A waiting thread is a thread that is waiting to be triggered into a non-active state.

[0014] After placing 12 the database change on top of the queue together with any associated interested listeners, a determination is made 14 as to whether the number of existing transaction service threads is greater than zero. If not, a new transaction service thread 16 is added. Otherwise, in one configuration, a determination is made 18 as to whether the number of existing transaction service threads is less than a dynamically determined optimum number of transaction service threads. A "dynamic determination" of a parameter utilizes current values of variables upon which the parameter is dependent in making the determination. A different value may be found for the parameter each time the parameter is determined. For example, the optimum number of transaction service threads is dependent upon a ratio of an arrival rate of database changes to the queue divided by a service time of items removed from the queue, and in one configuration, the optimum number of transaction service threads is equal to this ratio.

[0015] If determination 18 is that the number of transaction service threads is not less than optimum, a check 24 is performed to determine whether there are any threads waiting. If so, a waiting thread is changed 26 to non-active state, or if not, no change is made 20 to the number or state of transaction service threads. Otherwise, a further determination is made 22 to as to whether the number of transaction service threads is less than a predetermined maximum. (The predetermined maximum may, for example,

be "hard coded" or be provided as input during initialization of process 10.) If the number is determined to be less than the predetermined maximum, a new thread is added 16. Otherwise, if any transaction service threads are waiting 24, the state of one of the waiting transaction service threads is changed 26 to non-active. If no transaction service thread is waiting, no change 20 is made to the number or state of the transaction service threads.

[0016] In each case in which a new transaction service thread is added 16, that thread is placed into a non-active state 28. From this state, the new transaction service thread attempts to service the queue.

[0017] Referring to Figure 2, a flow chart 30 of one configuration of a transaction service thread that has just been placed in a non-active state 28 is shown. After entering non-active state 28, a determination is made 32 as to whether the total number of threads is greater than the predetermined maximum. If so, the thread is removed 34 (i.e., killed.) Otherwise, a test is performed to determine 36 whether the total number of transaction service threads is greater than a dynamically determined optimum number, determined in the same manner as described above at determination 18 in Figure 1. (These checks are necessary because additional transaction service threads may have been added by queuing process 10 since the thread entered 28 the non-active state, or while a transaction was being processed 38, 42, 44, 46, 48, 54, 56, 58, 60, 50, 52 by the thread.) If the determination is made that the number of transaction service threads is greater than optimum, the thread is removed 34. (In this configuration, it is thus possible, under certain conditions, for a thread placed 28 in a non-active state by process 10 to terminate itself 34 before it has checked the queue or

processed any database changes.) Thus, a thread is removed conditioned on there being more than the lesser of a predetermined maximum number or a dynamically determined optimum number of transaction service threads present.

[0018] If the total number of transaction service threads is less than the predetermined maximum and not greater than optimum, the thread determines 38 whether the queue size (i.e., the number of database changes) is greater than zero. If not, the thread enters 40 a waiting state. Waiting transaction service threads are ready and waiting to be triggered to a non-active state at step 26 of process 10, as shown in Figure 1.

[0019] If the queue size is greater than zero, the transaction service thread enters 42 an active state to process a database change in the queue. In the active state, the transaction service thread removes 44 the bottom element from the queue. The bottom element is the oldest remaining database change that is in the queue at the time of removal. In one configuration, the corresponding set of interested listeners is also removed, if any associated interested listeners were placed on the queue along with the database change. Any such listeners are then notified 46 that changes to the database indicated by the queued database change have begun. If there is no set of listeners corresponding to the removed database change, no listeners are notified at 46.

[0020] A database locking mechanism may be used to restrict access to the data source (e.g., a database server). The choice of a suitable locking mechanism is a design choice that may be made by one of ordinary skill in the art. In one configuration in which a database locking mechanism is

used, a determination is made 48 as to whether the transaction service thread is able to obtain the locks (i.e., the permission) necessary to perform the transaction specified in the database change that was removed from the bottom of the queue. If not, the transaction is not performed, any locks that may have been obtained are released 50 to permit other transactions to proceed. If any listeners were also taken from the queue, these listeners are notified 52 of the completion state of the database change, i.e., a completion state corresponding to "failure" is sent to the listeners. The transaction service thread then re-enters the non-active state 28 and attempts to process another database change from the bottom of the queue.

[0021] If it is determined 48 that locks can be obtained for the transaction, the locks are obtained 54 and the database transaction is begun 56, such as by performing any necessary initialization steps. For example, in one configuration, the transaction service thread runs in a client computing apparatus separate from a database server machine, and a session must be set up before the transaction can take place. After initialization is complete, the transaction is performed 58 and committed 60, and any necessary finalization is performed. All locks previously obtained are then released 50, and any listeners removed from the bottom of the queue are notified 52 of the completion state of the database changes. Usually, a "success" message is sent to the listeners in this case, but "failure" would be sent if the transaction could not be completed and/or committed for any reason. The transaction service thread then re-enters the non-active state 28 and attempts to process another database change from the bottom of the queue.

[0022] It is significant that, in Figure 1, threads may be added 16 to the total number of transaction service threads, while in Figure 2, threads may be removed 34 from the total number of transaction service threads. Thus, it is possible to dynamically change the optimum number of threads based upon a ratio of arrival rate to service rate, and to dynamically change an optimum size of the threads. For example, while at one time it may be determined that eight transaction service threads is an optimum number, after a time it may be determined that five transaction service threads is optimum. By providing for the addition 16 of threads in Figure 1 and the removal 34 of threads in Figure 2, the total number of transaction service threads can be kept to a minimum number necessary to successfully service the transaction queue, or to any other appropriate number.

[0023] In one configuration of the present invention, object-oriented programming techniques are used for implementation. A TaskedQueue is provided as an expanded thread pool that includes the concept of listeners and that can find an optimum number of service threads to perform work. Each service thread takes Java Runnables (a class that can be run in a separate thread) or Abortables (a class that can be run in a separate thread and safely aborted). Optimizations provide that, if possible given a specified maximum, the TaskQueue will use the minimum number of threads (system resources) necessary to continually service the queue. This optimization prevents the queue from being over serviced or allowed to grow faster than it can be serviced. A TransactionQueue is provided as a specialized TaskedQueue in which each service thread is configured for handling database transactions.

[0024] An example of a computing apparatus 200 configured to operate in accordance with the above description is represented in Figure 3. Cabinet 202 houses a central processing unit (CPU) and associated random access memory (neither of which are shown separately in Figure 6). In addition to random access memory, the CPU also communicates with other memory storage devices, for example, floppy disk drive 204, hard disk drive 206, and CD-ROM drive 208. A keyboard 210 is also provided. In one configuration, computing apparatus 200 is a database server. Computing apparatus system 200 in one configuration also includes a plurality of devices such as printers 212, 214, and 216 that can be used for printing data. In one configuration, users enter database changes at the CPU housed in cabinet 202, for example via keyboard 210. In another configuration, users communicate database changes to the CPU housed in cabinet 202 via terminals 218, 220, and 222, which can be computers running database client software. In one configuration, machine readable instructions configured to control server 202 CPU and/or computers 218, 220, and 222 to execute the steps and procedures described herein are recorded on one or more media 224, for example, one or more floppy diskettes or CD-ROMS. It is important to note that "computing apparatus" as used herein is intended to be construed in a general sense, in that the term may, in general, refer to a computer network 200 such as illustrated in Figure 3, or to an individual computer such as represented by 202, 218, 220, or 222. Also, the term "machine readable medium or media having instructions recorded thereon" is used in recognition of the fact that many computer programs or collections of programs intended to instruct a computing apparatus to perform a task are supplied on more than

a single diskette, CD-ROM, or other physical medium, or even on combinations of different types of physical media (e.g., a diskette and a CD-ROM). Whether a program is supplied on a medium or on media may depend upon the number and length of the instructions needed on the medium or media or upon design choices made by one of ordinary skill in the art.

[0025] It will thus be seen that configurations of the present invention efficiently handle the overhead of performing database transactions as well as efficiently manage resources related to the database. In particular, the queuing mechanism and the thread management mechanism reduces the likelihood that the database or database server will be "stressed" or "bogged down" with excessive numbers of transactions. In addition, the queuing of interested listeners along with database changes can be done automatically, without requiring users of a database to assume the overhead of managing their own transactions. Configurations of the present invention thus present a database resource to users with reduced likelihood that users will be able to overwhelm database capabilities, and without requiring users to configure each database manipulation as a separate transaction.

[0026] The description of the invention is merely exemplary in nature and, thus, variations that do not depart from the gist of the invention are intended to be within the scope of the invention. Such variations are not to be regarded as a departure from the spirit and scope of the invention.